Vehicle Detection Project

The goals / steps of this project are the following:

1. Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
2. Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
3. Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.
4. Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
5. Run your pipeline on a video stream (start with the test_video.mp4 and later implement on full project_video.mp4) and create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.
6. Estimate a bounding box for vehicles detected.

Note on Output Images and Videos:

- The output image of each test image in the 'test_images' folder is in the 'outputs' folder, this folder is in 'output_images' folder.
- The output image of HOG features in each color space on a car and a not car image is in folder 'hog_images', this folder is in 'output_images' folder
- The output of test on 'test_video.mp4' is 'test_output.mp4' and is in the folder 'output_images'. I tested my pipeline on 'project_video.mp4', the output of this is 'project-output.mp4' and is stored in 'output_images' folder.
- The output heatmap image on each test image is in 'heatmap' folder, this folder is in 'output_images' folder.
- The output image having bounding boxes on each test image is in 'bboxes' folder, this folder is in 'output_images' folder.

All code is in file CarND-Vehicle-Detection.ipynb

Histogram of Oriented Gradients (HOG)

1. Explain how (and identify where in your code) you extracted HOG features from the training images.

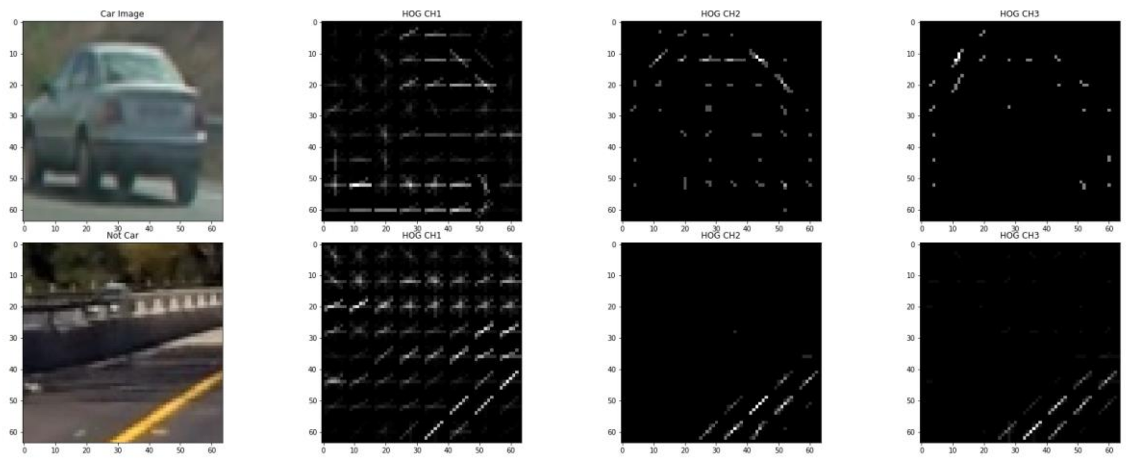
The code for this step is scattered. I have initialized parameters like ‘orient’, ‘pixel_per_cell’ and ‘cell_per_block’ is in 2nd code cell of the IPython notebook.

I have implemented ‘convert_color’ function to convert image in 3rd code cell and ‘skimage.hog()’ function in 4th code cell.

I started by reading in all the `vehicle` and `non-vehicle` images. The code for reading ‘car’ and ‘not_car’ image is in 5th code cell. Here is an example of one of each of the `vehicle` and `non-vehicle` classes:

I then explored different color spaces and different `skimage.hog()` parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`). I grabbed random images from each of the two classes and displayed them to get a feel for what the `skimage.hog()` output looks like. The code to extract hog features of each channel of each color space is in 6th code cell.

Here is an example using the `YCrCb` color space and HOG parameters of `orientations=8`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)`:



2. Explain how you settled on your final choice of HOG parameters.

I tried various combinations of parameters and color space. I tried on six color space (RGB, HLS, HSV, LUV, YUV, YCrCb), output of hog image on each channel of each color space is saved in 'Hog Images' folder. This folder is in 'output_images'. From images, it seems YCrCb color space gives most unique features for vehicle and non-vehicle images. So, I settled with YCrCb 'color space', no. of orientations = 8, pixel_per_cell = (8, 8) and cell_per_block = (2, 2) and hog channel = ALL, I will take hog along each channel of image and concatenate them.

3. Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

I trained a linear SVM using only HOG features and using around 2000 images, around 1000 images of class 'vehicle' and 1000 images of class 'non-vehicle'. The code to extract HOG features from each image in training data includes function 'extract_features' and is implemented in 7th code cell of the IPython notebook. The code to read images, extract features and training classifier is in 8th code cell of ipython notebook. After extracting features of images in training data, I then normalized the data using sklearn.preprocessing's StandardScaler. After normalizing, I split my data into training and test using train_test_split of sklearn.model_selection. I have used Linear SVM as classifier. I attained accuracy of 96.44%.

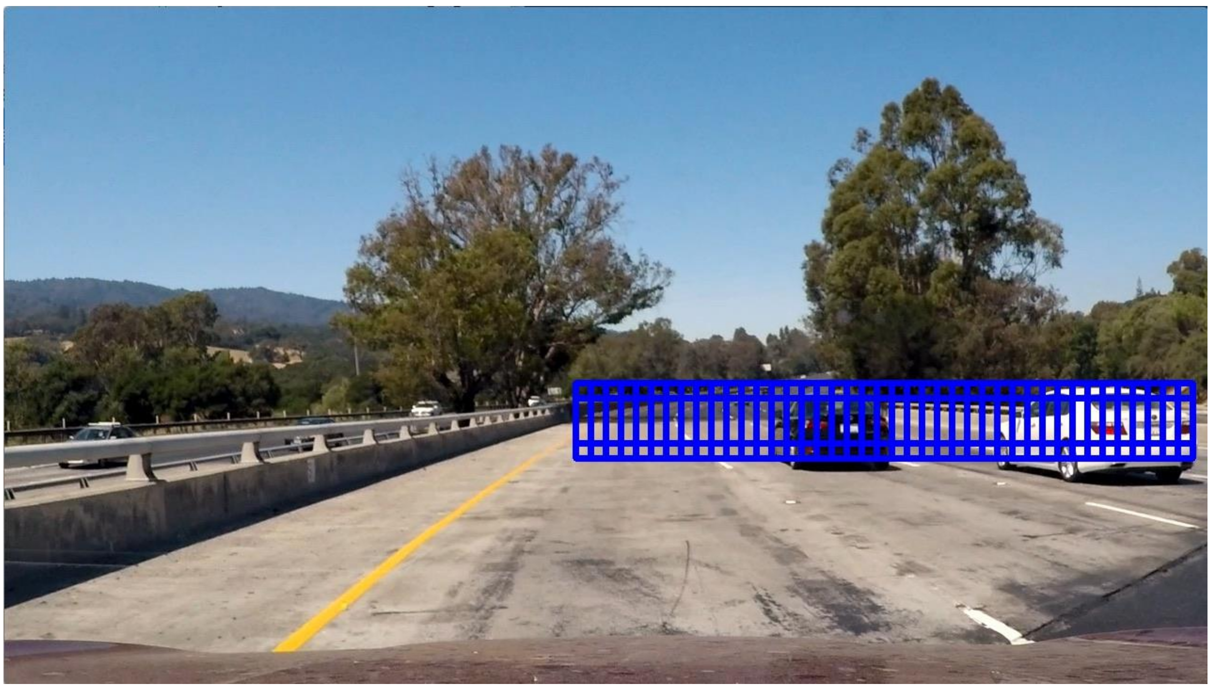
Sliding Window Search

1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

To identify vehicles in an image, I will use a sliding windows approach to detect vehicles. So for this we have created 'find_cars' function, this function is implemented in 9th code cell. In this, I start with reading 'svc' and 'scaler' model using load function. Then, I shrink the image to along y axis to concentrate only on 100 pixels. I then extract hog feature only once along all channel of image. I then loop through only single window size i.e. (64, 64) (more than one window size can be used, I have used one to reduce computation in each iteration). I then initialize various parameters to slide window. I'm using overlap of 0.75 or cells_per_step = 2. I start sliding window along x axis at half (to reduce computation and focus only on region Of Interest).

My aim was to reduce computation as much as possible while maintaining accuracy. Thus, I settled with 0.75 overlap and window size (64, 64).

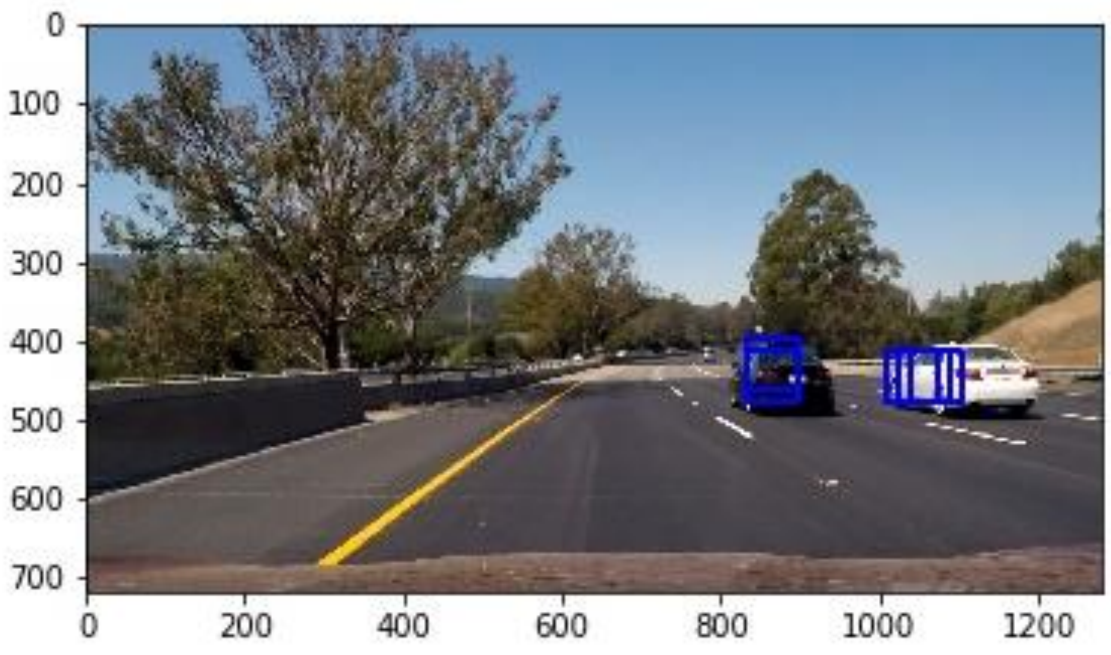
This I what I get after sliding window on a test image.



2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

Ultimately I searched on single window size i.e. (64, 64) using YCrCb 3-channel HOG features which provided a nice result. For each frame in pipeline, my focus was to reduce computation as much as possible while maintaining accuracy. To achieve this, I take HOG features along 100 pixels in y axis only once. Then, I sub-sample the HOG feature for each window, to predict car in it.

Here is an example output image:



Video Implementation

1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

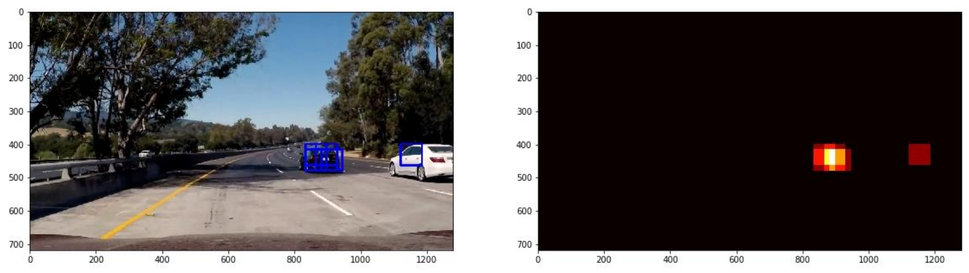
Here's a link to my test_output.mp4, project-output.mp4.

2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

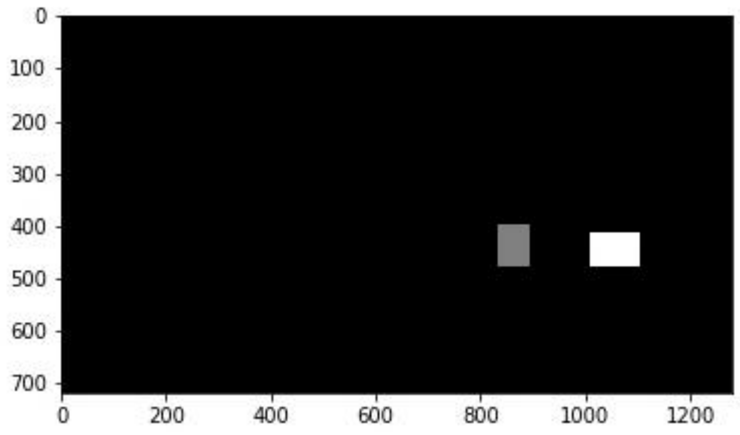
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap. I didn't threshold the heatmap as I'm not using multiple window size. I think thresholding will be useful when we have multiple window size. Also, not using thresholding was to minimize computation. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.

Here's an example result showing the heatmap from a series of frames of video, the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid on the last frame of video:

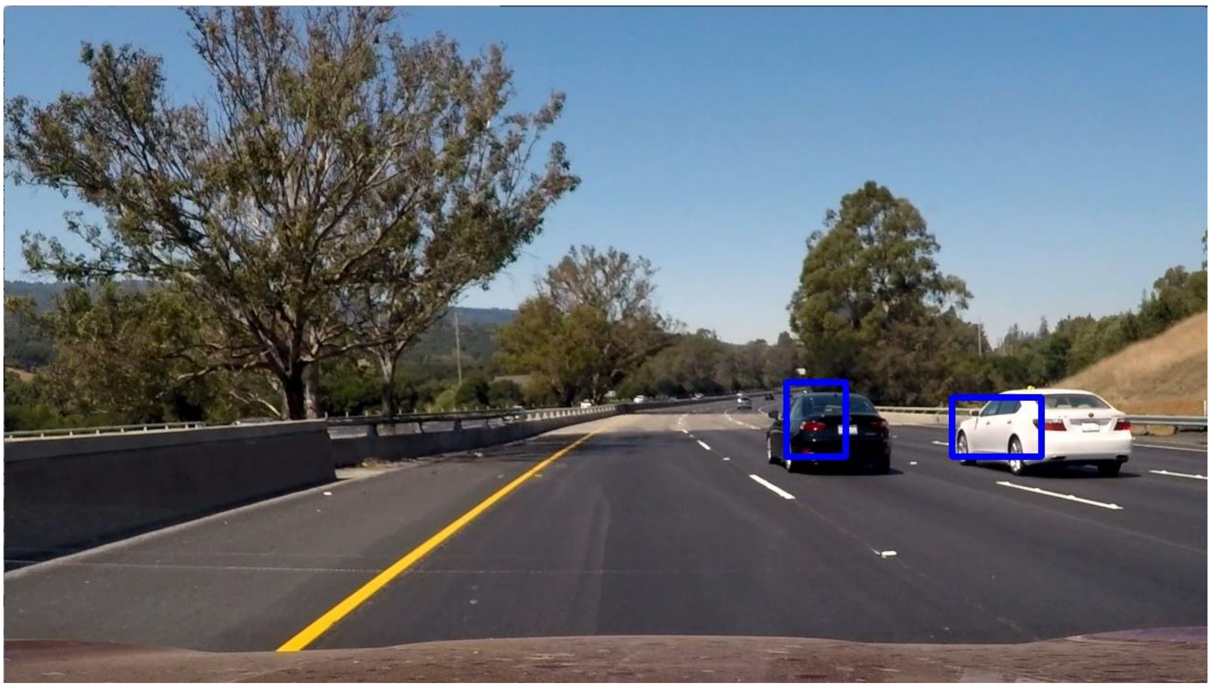
Here are six frames and their corresponding heatmaps:



Labels image visualization:



Here is an ouput image on a test image:



Discussion

1. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

I'm very happy with my implementation and the result I got. The main issue was the processing time, focus was to reduce it as much as possible. I think there's need to work further to reduce the processing time. I took 19 mins to process a 50 seconds video i.e. 1260 frames.

I think running a neural network will better predict car and resuce processing time, I have heard about U-Net to detect vehicles, looking forward to work on it.

Also, I think using a LIDAR or RADAR can very helpful in detecting vehicle/object/pedestrian/trees etc.